

Netcore应用

dotnet \*.dll --urls "http://\*:5001;"

dotnet \*.dll -ip 127.0.0.1 -port 5026s

dotnet WebApiCore.dll --urls=http://\*:5026 --ip=127.0.0.1 --port=5026

dotnet WebApiCore.dll --urls=http://\*:5027 --ip=127.0.0.1 --port=5027

dotnet WebApiCore.dll --urls=http://\*:5028 --ip=127.0.0.1 --port=5028

dotnet WebApiCore.dll --urls=http://\*:5029 --ip=127.0.0.1 --port=5029

dotnet WebApiProduct.dll --urls=http://\*:5126 --ip=127.0.0.1 --port=5126

dotnet WebApiProduct.dll --urls=http://\*:5127 --ip=127.0.0.1 --port=5127

dotnet WebApiCoreGateway.dll --urls=http://\*:6299 --ip=127.0.0.1 --port=6299

dotnet WebApiAuthentication.dll --urls=http://\*:7200 --ip=127.0.0.1 --port=7200

dotnet IdentityServerCore3.dll --urls=http://\*:6200 --ip=127.0.0.1 --port=6200

IdentityServerCore3.dll

Consul.exe 启动指令: Consul.exe agent -dev

Nginx启动指令：

start nginx.exe

./nginx -s stop  
./nginx -s quit  
./nginx -s reload

2.重新加载配置文件： 当 ngin x的配置文件 nginx.conf 修改后，要想让配置生效需要重启 nginx，使用-s reload不用先停止 ngin x再启动 nginx 即可将配置信息在 nginx 中生效，如下： ./nginx -s reload

dotnet WebCoreTest3.1.dll --urls=http://\*:6001 --ip=127.0.0.1 --port=6001

dotnet IdentityServerCore3.1.dll --urls=http://\*:5005--ip=127.0.0.1 --port=5005

https://localhost:5001/.well-known/openid-configuration

dotnet MvcClient.dll --urls=http://\*:5006

Redis基本指令

八种类型

string / hash / list /set /zset

List 分页第一页 热数据 数据库中经常查询的数据 很少修改的数据

Set 去重数据

Zset 带排序的权重数据

服务端

redis-server.exe redis.windows.conf

客户端

redis-cli -h 127.0.0.2 -p 6379 -a 123456

启动客户端显示中文字体：

redis-cli -h 127.0.0.1 -p 6379 -a 123456 --raw

Redis 集群搭建最少6个节点

主从复制+分片

主从复制：当一个集群中的主节点挂了，则有从节点代替

分片：解决所有的读取不会所有的数据读取一台集群或者一个主节点

启动集群指令

Linux ./redis-trib.rb create --replicas 1 127.0.0.2:6379 127.0.0.2:6380 127.0.0.2:6381 127.0.0.2:6382 127.0.0.2:6383 127.0.0.2:6384

Windows redis-trib.rb create --replicas 1 127.0.0.2:6379 127.0.0.2:6380 127.0.0.2:6381 127.0.0.2:6382 127.0.0.2:6383 127.0.0.2:6384

注意127.0.0.2 ip时失败

redis-trib.rb create --replicas 1 127.0.0.1:6379 127.0.0.1:6380 127.0.0.1:6381 127.0.0.1:6382 127.0.0.1:6383 127.0.0.1:6384

客户端

redis-cli -h 127.0.0.1 -c -p 6379 -a 123456

Redis-cli -h 127.0.0.1 -c -p 6380 -a 123456

Redis-cli -h 127.0.0.1 -c -p 6381 -a 123456

Redis-cli -h 127.0.0.1 -c -p 6382 -a 123456

Redis-cli -h 127.0.0.1 -c -p 6383 -a 123456

Redis-cli -h 127.0.0.1 -c -p 6384 -a 123456

Flushall

Cluster reset

redis-trib.rb check 127.0.0.1:6379

官方是推荐使用redis-trib.rb fix 来修复集群。通过cluster nodes看到7001这个节点被干掉了。可以按照下面操作进行修复

[root@slave2 redis]# redis-trib.rb fix 127.0.0.2:6380

修复完成后再用check命令检查下是否正确（查看别的节点）

[root@slave2 redis]# redis-trib.rb check 127.0.0.2:6380

只要输入任意集群中节点即可，会自动检查所有相关节点。

可以查看相应的输出看下是否是每个Master都有了slots。

如果分布不均匀那可以使用下面的方式重新分配slot:

[root@slave2 redis]# redis-trib.rb reshard 127.0.0.2:6379

集群搭建遇到问题

port 7000 //端口7000,7002,7003

bind 本机ip //默认ip为127.0.0.1 需要改为其他节点机器可访问的ip 否则创建集群时无法访问对应的端口，无法创建集群

daemonize yes //redis后台运行

pidfile /var/run/redis\_7000.pid //pidfile文件对应7000,7001,7002

cluster-enabled yes //开启集群 把注释#去掉

cluster-config-file nodes\_7000.conf //集群的配置 配置文件首次启动自动生成 7000,7001,7002

cluster-node-timeout 15000 //请求超时 默认15秒，可自行设置

appendonly yes //aof日志开启 有需要就开启，它会每次写操作都记录一条日志

syslog-ident = redis  
Invalid argument during startup: unknown conf file parameter : cluster-config-file

我一开始以为是redis的windows服务版本不支持这个配置选项,网上各种找和查,最后才知道原来redis的配置项前面不能有空格

关于启动redis集群时：

**[ERR] Node 172.168.63.202:7001 is not empty. Either the nodealready knows other nodes (check with CLUSTER NODES) or contains some**

的问题

解决办法：

1)将每个节点下aof、rdb、nodes.conf本地备份文件删除；   
2)172.168.63.201:7001> flushdb #清空当前数据库(可省略)

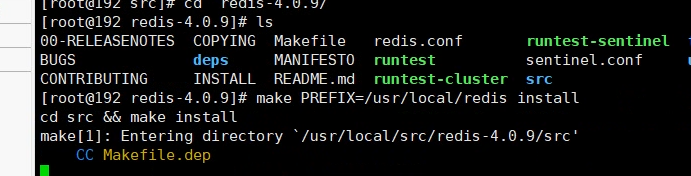
dotnet ConsoleAppServer.dll --ip=127.0.0.1 --port=998

秒杀测试指令

dotnet ConsoleAppNetCore3.dll 1 56

Redis Lunix安装

make PREFIX=/usr/local/redis install



cd /usr/local/redis/bin/

./redis-server redis.conf

./redis-cli -h 127.0.0.1 -p 6379

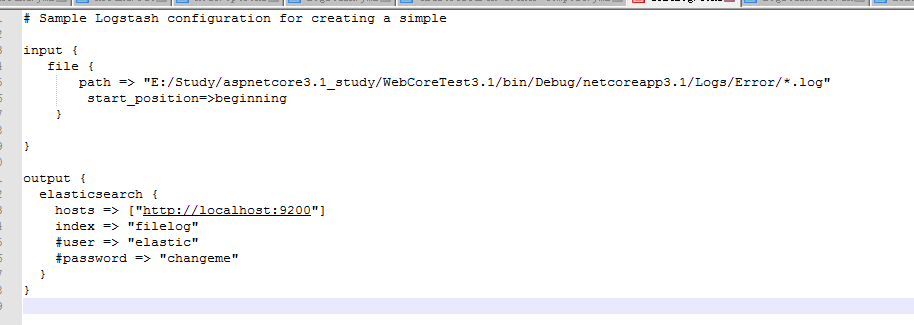
# ELK Elasticsearch Logstash Kibana

一、Logstash 启动配置：

1、指令：logstash.bat -f filelog.conf

Filelog.conf配置内容

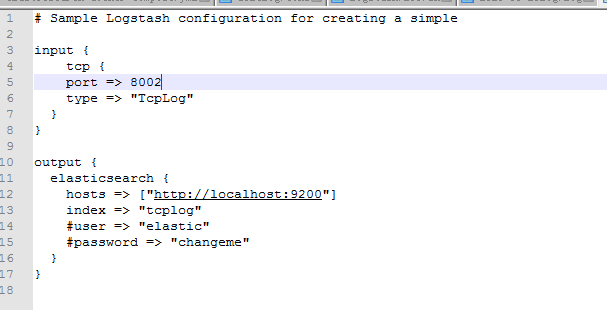
注意点：path路径斜线是 “/” 不是 “\”



2、Tcplog.conf配置内容

dotnet WebCoreTest\_ELK\_Logstash\_TCP.dll --urls=http://\*:8003 --ip=127.0.0.1 --port=8003

logstash.bat -f tcplog.conf

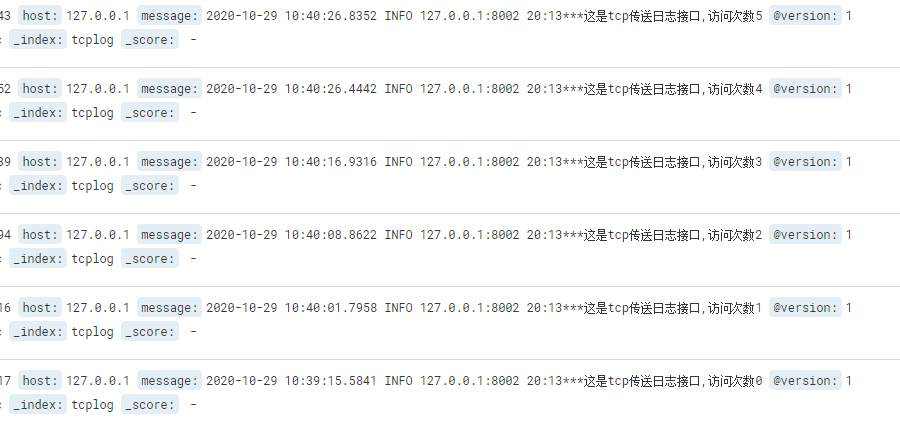


注意点：core项目配置nlog.config时配置tcp协议

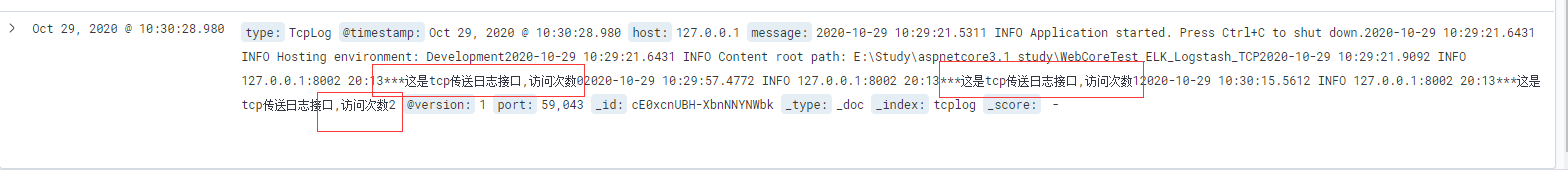
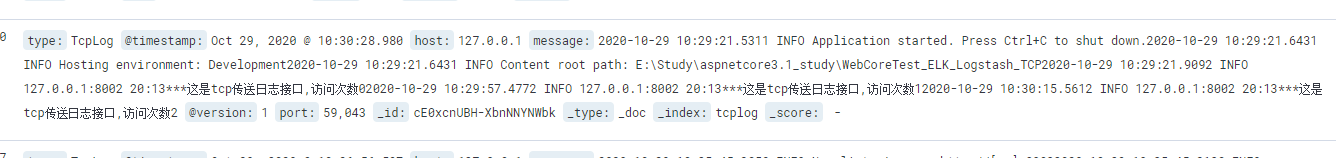
<target xsi:type="Network" name="logInfo" address="tcp://127.0.0.1:8002" layout="${longdate} ${uppercase:${level}} ${message}" keepConnection="false"/>



当属性keepConnection="false"，可以持续写入，



当设置true时，只能关闭应用后一次性写入，我第一次采坑点就是没有配置<target xsi:type="Network" name="logInfo" address="tcp://127.0.0.1:8002" layout="${longdate} ${uppercase:${level}} ${message}" />导致只能关闭应用时候一次性写入。我个人猜测是tcp端口持续暂用导致，估计八九不离十（我猜测是在本地运行core项目中应该是只能写进去一次，8002被占用，logstash应该调用不到，导致日志暂时写不进去，当core项目停止后，logstash能通过tcp传送成功）

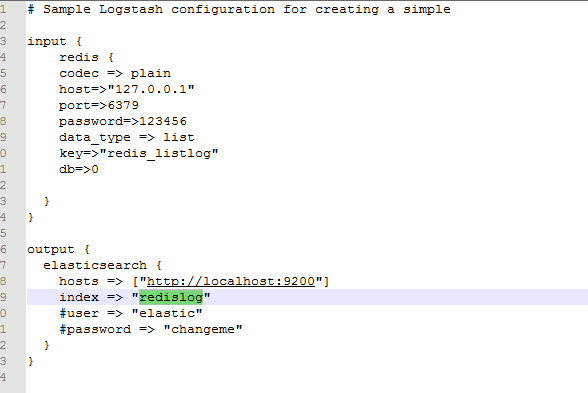


3、Redis 方式写入

logstash.bat -f redislog.conf

配置好之后按照顺序重启：redis-- es ---logstash--kibana 客户端写入正常拉取

（大数据高并发情况下日志可以采取存储到redis list中，然后配置logstash去拉取数据，logstash拉取数据后会删除，所以不用担心内存不足问题

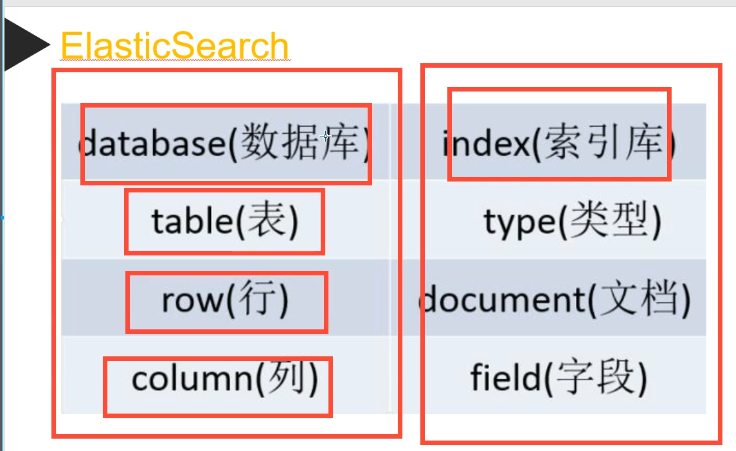
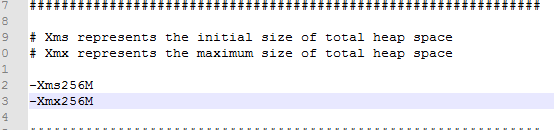


### ElasticSearch

Elasticsearch配置：

Config/jvm.options:

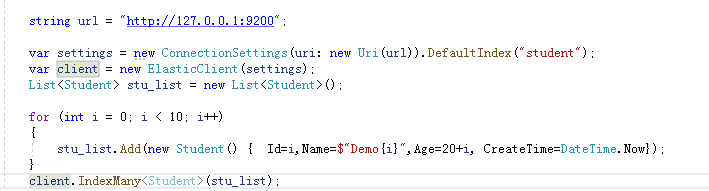
-Xms/Xmx默认是1g,改成256M



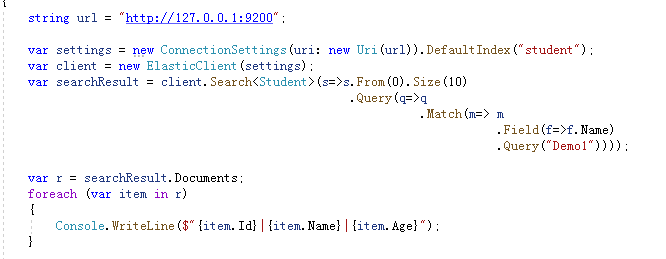
.NetCore调用需要引用NEST包

官方文档：https://www.elastic.co/guide/en/elasticsearch/client/net-api/current/index.html

插入es数据库代码



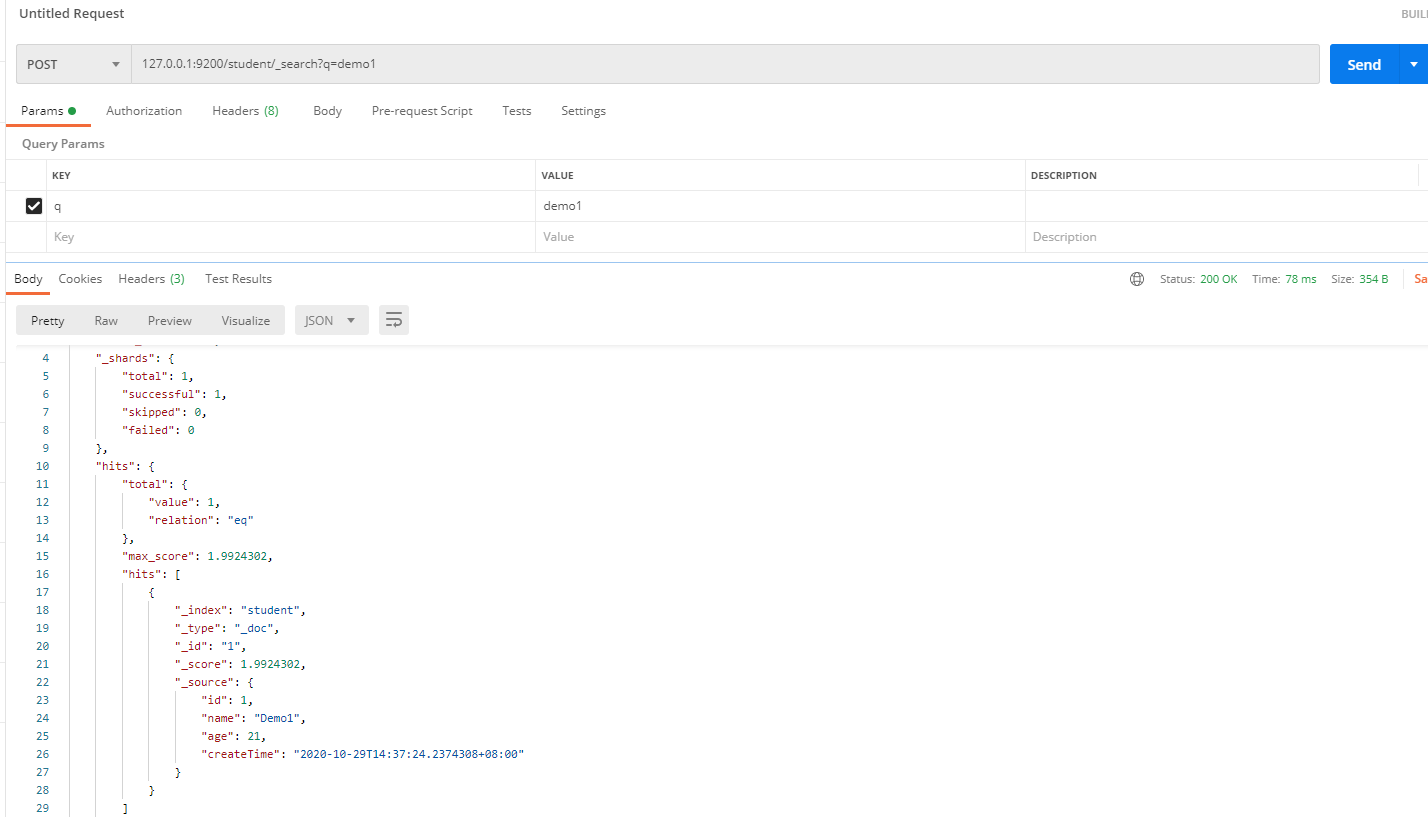
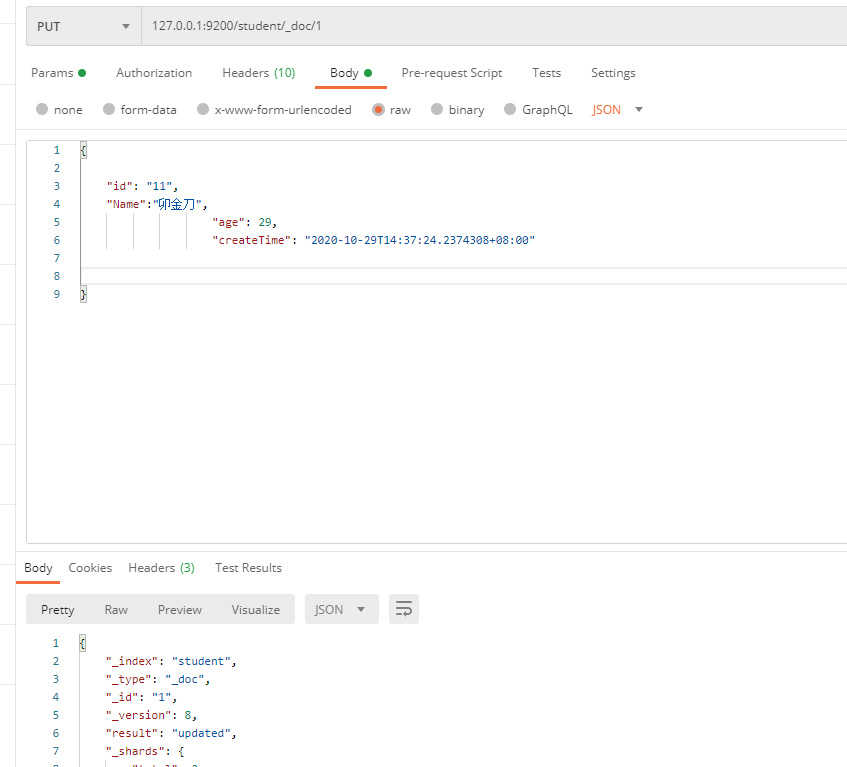
获取代码：



Postman操作

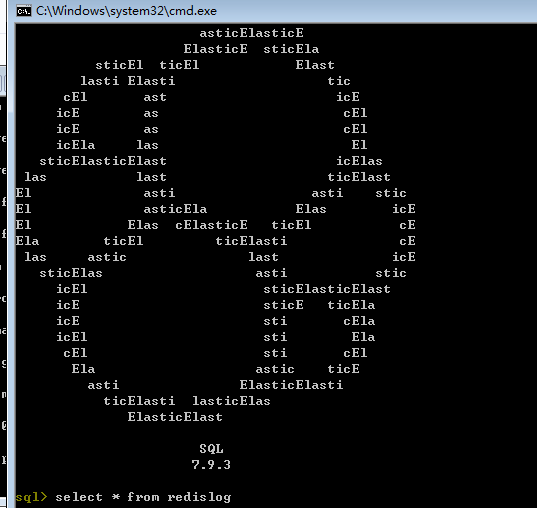
此篇博客可以参考https://blog.csdn.net/u013613428/article/details/56484794

官网文档https://www.elastic.co/guide/en/elasticsearch/reference/current/rest-apis.html



使用sql操作

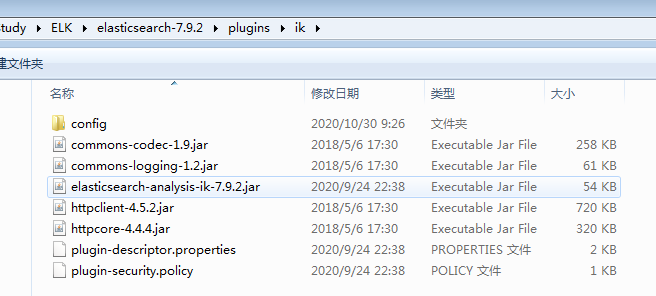
Bin/elasticsearch-sql-cli.bat



IK分词器（插件中文分词）

\elasticsearch-analysis-ik-7.9.2\bin\文件加下的分词器文件copy放到

elasticsearch下面的一个\plugins文件夹下建立一个新的文件夹\ik\文件夹下



案例：

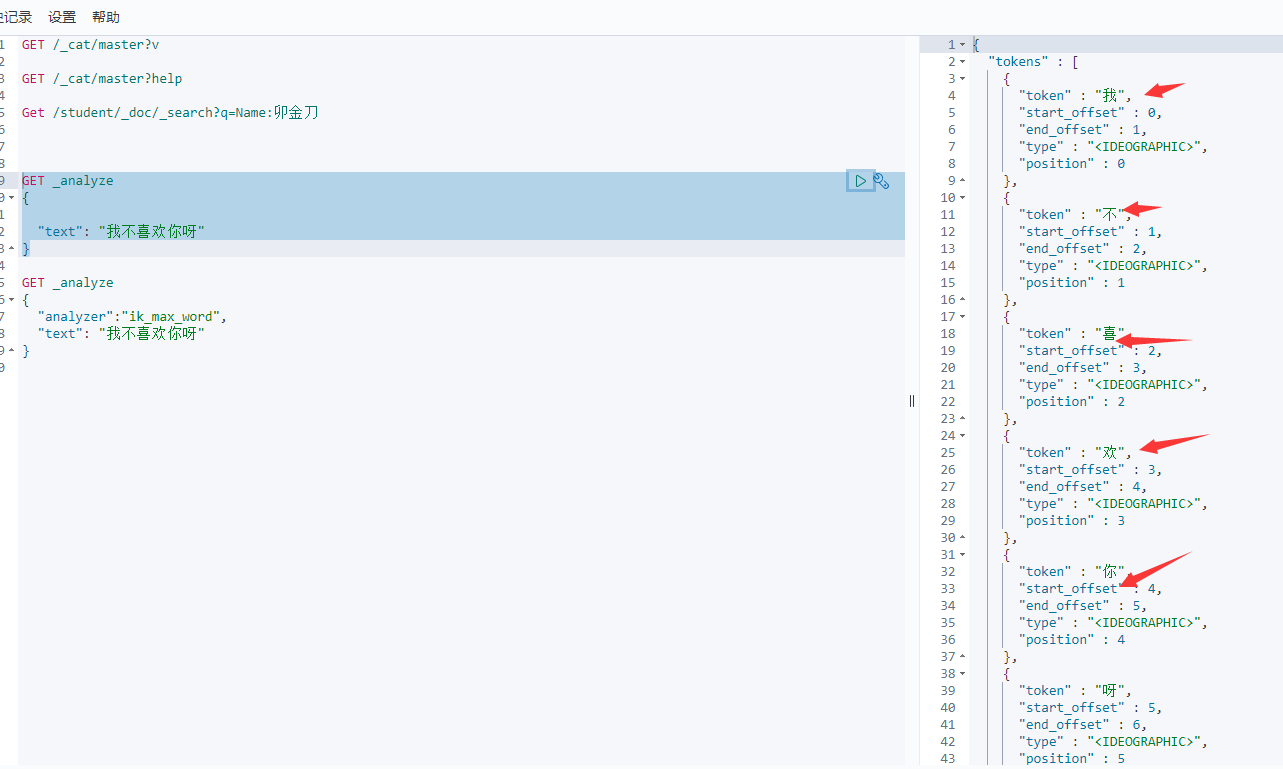
未使用分词器：

GET \_analyze

{

"text": "我不喜欢你呀"

}



使用分词器：

GET \_analyze

{

"analyzer":"ik\_max\_word",

"text": "我不喜欢你呀"

}



1. Kibana

使用前先安装java环境包

查询操作

Get /student/\_doc/\_search

Get /student/\_doc/\_search?q=Name:卯金刀

# Docker

命令：

docker --version 查看当前版本

docker ps 查看当前运行的容器

docker images 查看本地所有的镜像

docker run 运行一个容器

docker pull 用来拉取一个镜像到本地

1 docker logs 容器名称

2 docker attach 容器名称或容器ID

3 docker exec -it 59ec /bin/bash (如果提示“/bin/bash 未找到”之类的 可以改成:docker exec -it 59ec sh,退出ctrl+d)

4 docker stop 容器ID或容器名

5 docker kill 容器ID或容器名

6 docker rm 容器ID或容器名

7 systemctl restart docker  
8 docker image prune 删除所有镜像  
9.docker container prune 删除所有容器

运行docker容器，后台运行

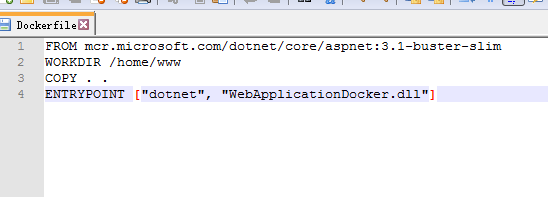
docker run -d -p 80:80 webapicore

docker stop 容器id

docker rm 容器id

部署CORE

1、上传Dockerfile文件



1. 项目上传后执行

cd /home/apitest/ 进入该项目目录

docker build -t webapicore -f Dockerfile . 注意一个英文的“.”不能少

docker build -t webmvccore -f Dockerfile .

1. docker run -d -p 5026:80 webapicore
2. docker run -d -p 5027:80 webapicore

docker run -d -p 8001:8001 webapicore

1. docker run -d -p 8001:80 webmvccore

查看端口监听情况

netstat -an | grep 8080

查找文件路径find / -name nginx.conf

查找文件夹 ls -l /usr/share/nginx/

编辑文件

vim nginx.conf

创建文件夹

mkdir filesname

nginx 文件挂载

docker run --name nginx -p 8086:80 -v /data/docker\_v/nginx/www:/home/configfile -d nginx

退出：先按【ESC】 再输入:q

常用操作指令http://www.51gjie.com/linux/1029.html

批量停用并删除全部容器

1.停用全部运行中的容器:

docker stop $(docker ps -q)

2.删除全部容器：

docker rm $(docker ps -aq)

3.一条命令实现停用并删除容器：

docker stop $(docker ps -q) & docker rm $(docker ps -aq)

docker stop $(docker ps -q) & docker rm $(docker ps -aq)

docker-compose命令安装镜像：

1、运行以下命令以下载Docker Compose的当前稳定版本：

sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

2将可执行权限应用于二进制文件：  
sudo chmod +x /usr/local/bin/docker-compose

3测试安装。  
docker-compose --version



1. 执行docker-compose命令

a先启动consul集群

cd /home/configfile/

docker-compose up

b在启动网站

cd /home/project/

docker-compose up

consule

拉取最新镜像：

docker pull consule

启动consul:

docker run -d -p 11500:8500 consul

consul 搭建集群

搭建集群前需要开通ip端口：tcp :8301,8302,8500,9500,10500,11500/udp:8301,8302

node1:

docker run -d --name=node1 --restart=always \

-e ’CONSUL\_LOCAL\_CONFIG={“skip\_leave\_on\_interrupt”:true}’ \

-p 8300:8300 \

-p 8301:8301 \

-p 8301:8301/udp \

-p 8302:8303 \

-p 8302:8302/udp \

-p 8400:8400 \

-p 8500:8500 \

-p 8600:8600 \

-h node1 \

consul agent -server -bootstrap-expect=3 -bind=0.0.0.0 -node=node1 \

-data-dir=/tmp/data-dir -client 0.0.0.0 -ui

node2

docker run -d --name=node2 --restart=always \

-e ’CONSUL\_LOCAL\_CONFIG={“skip\_leave\_on\_interrupt”:true}’ \

-p 9300:8300 \

-p 9301:8301 \

-p 9301:8301/udp \

-p 9302:8303 \

-p 9302:8302/udp \

-p 9400:8400 \

-p 9500:8500 \

-p 9600:8600 \

-h node2 \

consul agent -server -bind=0.0.0.0 \

-join=121.196.176.108 -node-id=$(uuidgen | awk '{print tolower($0)}') \

-node=node2 \

-data-dir=/tmp/data-dir -client 0.0.0.0 -ui

node3

docker run -d --name=node3 --restart=always \

-e ’CONSUL\_LOCAL\_CONFIG={“skip\_leave\_on\_interrupt”:true}’ \

-p 10300:8300 \

-p 10301:8301 \

-p 10301:8301/udp \

-p 10302:8303 \

-p 10302:8302/udp \

-p 10400:8400 \

-p 10500:8500 \

-p 10600:8600 \

-h node3 \

consul agent -server -bind=0.0.0.0 \

-join=121.196.176.108 -node-id=$(uuidgen | awk '{print tolower($0)}') \

-node=node3 \

-data-dir=/tmp/data-dir -client 0.0.0.0 -ui

node4

docker run -d --name=node4 --restart=always \

-e ’CONSUL\_LOCAL\_CONFIG={“skip\_leave\_on\_interrupt”:true}’ \

-p 11300:8300 \

-p 11301:8301 \

-p 11301:8301/udp \

-p 11302:8303 \

-p 11302:8302/udp \

-p 11400:8400 \

-p 11500:8500 \

-p 11600:8600 \

-h node4 \

consul agent -bind=0.0.0.0 \

-join=121.196.176.108 -node-id=$(uuidgen | awk '{print tolower($0)}') \

-node=node4 \

-data-dir=/tmp/data-dir -client 0.0.0.0 -ui

查看日志

docker logs -f node1

查看集群节点

docker exec -t node1 consul members

docker exec -t consul1 consul members

查看主从信息

docker exec -t node1 consul operator raft list-peers

docker exec -t consul1 consul operator raft list-peers

Nginx

nginx 挂载转发到consul集群

docker run -d -p 8089:80 -v /home/project/consulnginx/:/var/log/nginx/ -v /home/project/consulnginx/nginx.conf:/etc/nginx/nginx.conf --name consulnginx nginx

docker run -d -p 8088:80 -v /home/project/gatewaynginx/:/var/log/nginx/ -v /home/project/gatewaynginx/nginx.conf:/etc/nginx/nginx.conf --name gatewaynginx nginx

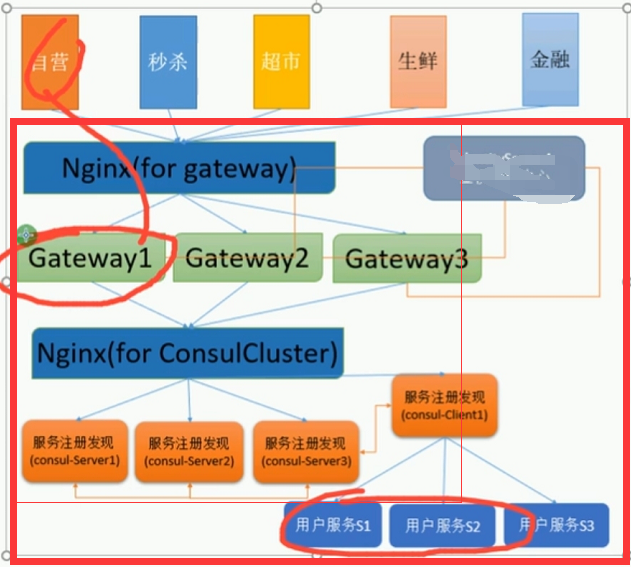
先启动consul集群，再启动项目

到此本次搭建的框架图：

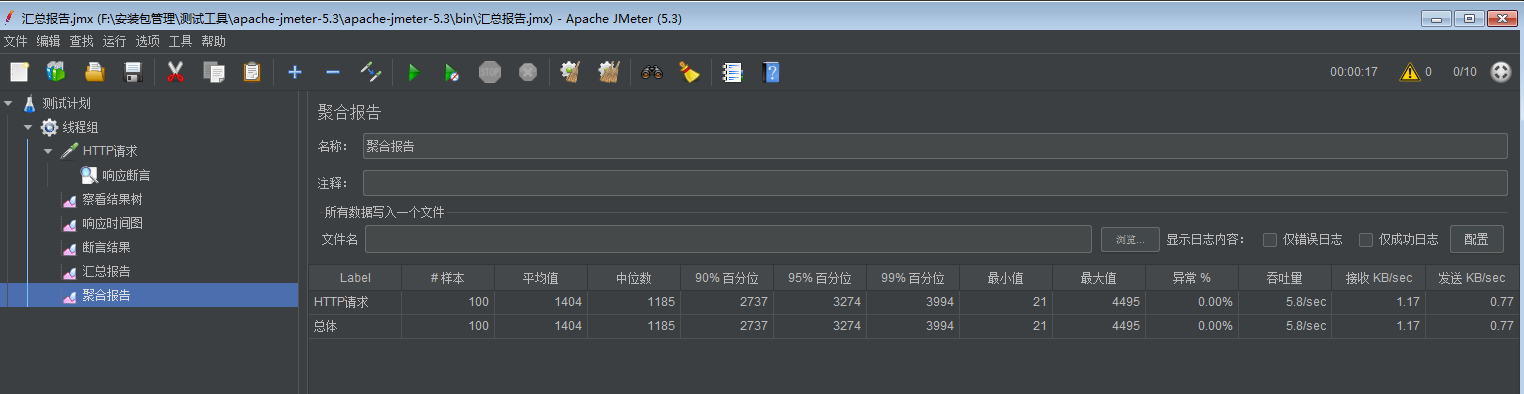
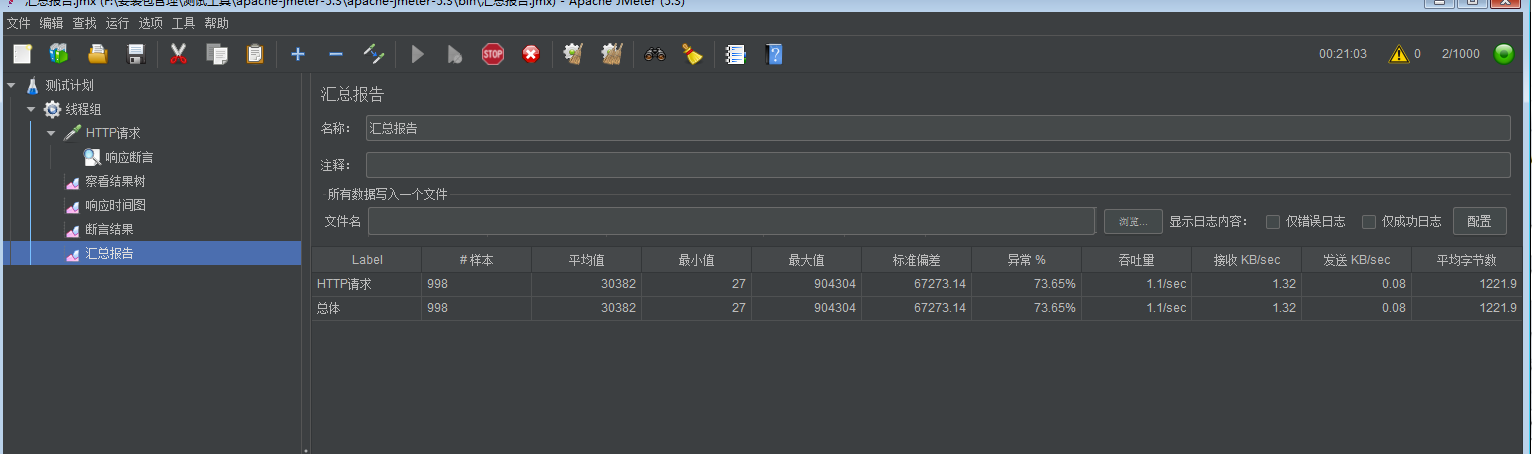
客户端— —》Nginx（gateway网关）——》网关（集群）——》Nginx（consulf服务发现与治理）——》consul(集群)——》webapi服务接口（集群）

：

|  |  |  |  |
| --- | --- | --- | --- |
|  | IP：Ports | 调用服务(请求网关) |  |
| 客户端 | 121.196.176.108:xxx | http://121.196.176.108:8088/pserver/products | http://121.196.176.108:8088/userver/user |
| Nginx（gateway网关） | 121.196.176.108:8088 |  |  |
| 网关（集群） | 121.196.176.108:6299/6399/6499 |  |  |
| Nginx（consulf服务发现与治理） | 121.196.176.108:8089 |  |  |
| consul(集群) | 121.196.176.108:8500/9500/10500/11500 |  |  |
| webapi\_user服务接口（集群） | 121.196.176.108:5026/5027/5028 |  |  |
| webapi\_product服务接口（集群） | 121.196.176.108:6026/6027/6028 |  |  |



JMeter 压测工具



解释一下这份报告：

样本：请求的次数，计算公式是线程数\*循环次数，如果线程组配置勾选了永远，那么就是你停止测试时实际发送的请求数

平均值：响应时间的平均用时，单位是毫秒。比如这里的平均响应时间是38毫秒

中位数：响应时间的中位数，单位是毫秒。

90%百分位：90%的响应时间小于该数值，单位是毫秒。这里有90%的响应时间小于22毫秒

95%百分位：含义和90%类似

99%百分位：含义和90%类似

最小值：本轮测试最小响应时间，单位是毫秒。

最大值：本轮测试最大响应时间，单位是毫秒。

异常%：本轮测试出现异常的请求比例。

吞吐量：可以理解为QPS，即是我们测试的接口处理请求的能力。比如这里是平均每秒可以处理2.2次请求

接收KB/Sec：响应数据的接收速率

发送KB/Sec：请求数据的发送速率